ARL-TR-7923 ● JAN 2017

US Army Research Laboratory

Python Scripts for Automation of Current-Voltage Testing of Semiconductor Devices (FY17)

by Bryan H Zhao and Randy P Tompkins

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-7923 ● JAN 2017

US Army Research Laboratory

Python Scripts for Automation of Current-Voltage Testing of Semiconductor Devices (FY17)

by Bryan H Zhao
Oak Ridge Institute for Science Education Research (ORISE)
Belcamp, MD

Randy P Tompkins
Sensors and Electron Devices Directorate, ARL

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
January 2017	Technical Report	

4. TITLE AND SUBTITLE
Python Scripts for Automation of Current-Voltage Testing of Semiconductor Devices (FY17)

5a. CONTRACT NUMBER
1120-1120-99

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Bryan H Zhao and Randy P Tompkins

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
US Army Research Laboratory
ATTN: RDRL-SED-E
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-7923

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Automation of device testing and analysis procedures is imperative in both research and industrial environments where automation improves employee efficiency. The “time sink” of manual device-testing procedures is reduced or eliminated through automation. This technical report includes scripts written in Python, version 2.7, used to either partially or fully automate our existing current-voltage (I-V) test setup at the US Army Research Laboratory. The I-V test setup currently includes a Wentworth PML 8000 manual probe station, an Agilent 4155C semiconductor parameter analyzer, and a Micromanipulator P200L semiautomatic probe station. In addition to the current version of the scripts, explanations of the different portions of the code are also provided for the user. Scripts used to set up this automated system are discussed in enough depth such that another user may operate the integrated system or build upon the existing code to improve its functionality and/or efficiency. This report assumes that the reader has some basic knowledge on the syntax of Python and thus is not a tutorial for the language. Additionally, we identify future software development tasks that could further expand the capabilities of the system.

15. SUBJECT TERMS
gallium nitride, GaN, diode, python, auto-prober, current-voltage testing

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Randy P Tompkins
Unclassified	Unclassified	Unclassified	UU	48	19b. TELEPHONE NUMBER (Include area code) 301-394-0015

Contents

List of Figures

1. Introduction

Semiautomatic probe stations are capable of probing hundreds of devices on semiconductor wafers in a relatively short period of time. Subsequently, integration of a semiautomatic probe station with semiconductor device testing instruments (i.e., current-voltage [I-V] or capacitance-voltage measurements), common for semiconductor devices such as power diodes, light-emitting diodes, solar cells, and transistors, can be done much faster than using manual probing stations.

Manual measurements can be very time consuming due to required user operation of probes, slow data transfer for old systems, and lack of optimized data parsing and analysis. One additional advantage of automation to device testing is the opportunity to test a large number of devices. Often, due to the significant time investment required by the user with manual operation, only a fraction of the total devices fabricated on a sample are actually tested.

In this technical report, we describe our new I-V testing system at the US Army Research Laboratory (ARL) that includes a micromanipulator P200L semiautomatic probe station fully integrated with our Agilent 4155C semiconductor parameter analyzer. Included in this technical report is a description of this system as well as the Python (version 2.7) scripts that were written to program the system and specific instructions to the user on how to operate the I-V testing system. In addition, we describe further ongoing efforts in software development to improve the existing system.

2. Micromanipulator P200L Semiautomatic Probing Station

The main functionality of the Micromanipulator P200L semiautomatic probe station comes from the ESP301 motion controller, which can move the stage in 3 dimensions, referred to as Axis 1, Axis 2, and Axis 3. It is important to note that the motion controller only moves the stage itself in 3 dimensions. It does not control the probes or stage rotation. It is important to note that the user must first attempt to manually align the samples, so that the horizontal and vertical lattices are perfectly aligned with the stage left–right movement. Due to these limitations, the P200L would be ideally used to probe a large amount of devices that are in a symmetric arrangement on the wafer. An example of a possible layout is shown in Fig. 1, where the devices are arranged in a square pattern with equal center to center distances in the vertical dimension and variable center-to-center distances in the horizontal direction.

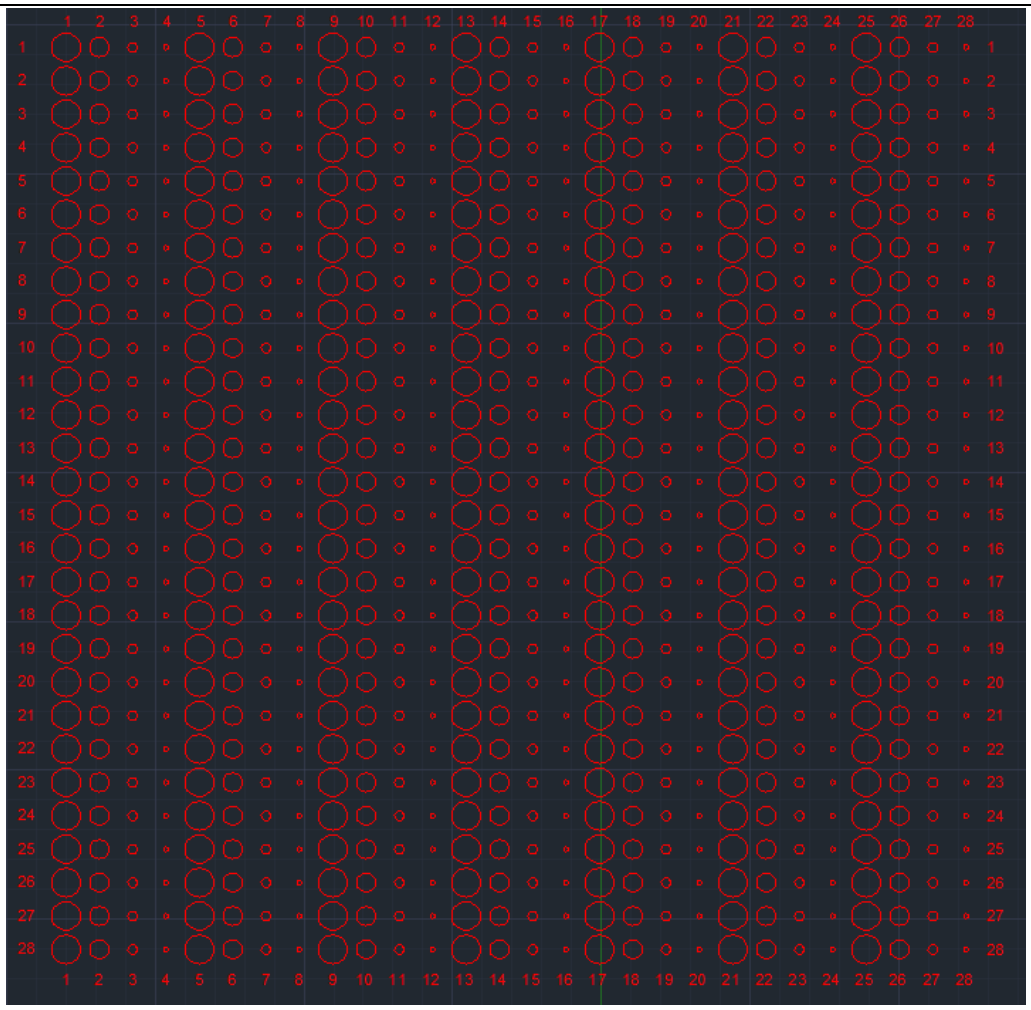

Fig. 1 Computer-aided design drawing of an example square symmetric device layout pattern: 28 × 28 devices, alternating in device size from 300, 200, 100, and 50 microns throughout each row

It is important for the reader to understand the orientation of the moving stage and to what the 3 axes refer to fully understand the axis movement portion of the code found in this technical report. When the user is facing the P200L, the following are true:

- Positive (+) Axis 1 is left.
- Negative (–) Axis 1 is right.
- Positive (+) Axis 2 is toward the user.
- Negative (–) Axis 2 is away from the user.
- Positive (+) Axis 3 is up.
- Negative (–) Axis 3 is down.

3. 4155C Setups

3.1 Wentworth Manual Probing Station

Because of the difficulty in programming the semiautomatic prober, we initially developed scripts to communicate with the semiconductor parameter analyzer to take basic I-V measurements of devices using our existing manual probe station. This code served as a prototype to the code used to integrate the 4155C with the P200L.

In the original setup (i.e., prior to any programming), the 4155C had to be controlled by the user at the instrument front panel, where the data collected were saved to a 3.5-inch floppy disk. This process was time consuming and slow due in large part to the slow data transfer rate. To decrease the time for each device measurement while using a manual probing station, an interface was created that eliminated usage of the front panel on the 4155C and writes the data files to a specified directory on the computer.

We expect that these scripts will continually be used with the manual probe station for special cases, for example, when a given mask set does not have a highly symmetric device arrangement or individual devices have a unique geometry that does not accommodate the P200L. This script also integrates 4 important measurements of both diodes and transistors: I-V, I_{ds}–V_{ds}, I_{ds}–V_{gs}, and OFF-state breakdown voltage into compact functions/cases. This allows the user to efficiently and quickly perform any of the 3 measurements with only one button.

Script 1 (HEMT_Diode_toComputer_withInterface_All3.py; see Section 4.1), at its highest level, is operated as a user interface (UI), which allows the user to more efficiently and effectively collect data measurements using the 4155C and the manual probe station. This script bypasses the original front panel operation of the 4155C and allows the user to set up parameters and run measurements with far fewer inputs.

This script has many failsafe/error-checking logic integrated into it that prevents the user from performing certain tasks out of sequence. Thus, there is a certain procedure to performing measurements with this script. A brief description of script 1 as well as guidance for the user is discussed.

The following is an example procedural run:

1) Select option 4 and choose a device (“hemt” or “diode”).
2) Select option 5, 6, or 7 to set the test parameters, depending on which tests will be performed.

3) Select option 1, 2, or 3 to actually execute the intended test with the specified test parameters.

Looking at the code for script 1, in lines 10 and 11, we establish connection with the 4155C through the PyVISA package in Python. This connection is formed through a general-purpose interface bus (GPIB) to USB plug between the instrument and the computer. In line 13, `del instr.timeout` is critical to the script running effectively. This line indicates to the 4155C to eliminate its timeout counter, which will automatically crash the script if a command takes longer than 5 s to execute. By eliminating this timer, the script will take as long as it needs to execute all of its commands, which will vary depending on the number of devices tested.

The basis for the UI is a continuous while loop that continues to repeat as long as the user wants to keep performing measurements. In lines 24 through 36, the basic UI is created by prompting the user for an input between numbers 1 and 9, which will then perform the intended task. The 9 different tasks this script is intended to perform are described individually in the following subsections.

Note: All GPIB commands referred to in this text can be found in each instrument's accommodating GPIB reference manual. The following text refers to the Agilent 4155C GPIB reference manual.[1]

3.1.1 Run the I-V Test

In this task, the script performs either an I-V (forward and reverse) measurement on a diode or an I_{ds}–V_{ds} measurement on a high electron mobility transistor (HEMT). In lines 42 through 52, the script is simply checking to see that a specific device type is selected and if the test parameters have been set. In lines 58 through 95, the script is writing a set of GPIB commands to the 4155C to perform a certain task depending on which device type is tested. In this case, this section is setting up the 4155C to perform an I-V curve measurement. These specific commands are selected based on how a user would manually set up an I-V measurement. For example, in line 60, `instr.write(":PAGE:CHAN:MODE SWEEP")` is simply telling the 4155C to set the channel mode to sweep. We do not discuss each command line by line here; please refer to the GPIB reference guide[1] for more information.

In lines 97 through 112, the script is setting test parameter values for each variable based on the stored values in the script, also depending on whether a diode or a HEMT is being tested (these said values are stored using command 5). Line 117, `instr.write(":PAGE:SCON:SING")`, executes the sweep or performs the measurement.

In the next section, between lines 121 and 157, the script is formatting the output data so that they can written to a text file in a specific way so that data analysis/parsing can be easily performed. In this section, the most important lines are lines 126 and 128, `I_data = instr.query_ascii_values(":DATA? 'ID' ")` and `I_data = instr.query_ascii_values(":DATA? 'ID' ")`, respectively. These lines are directly telling the 4155C to store the ASCII data for I_D and V_D into lists in the script, which can then be parsed and formatted into any form the script writer wants. The code in the rest of this section is data formatting. Lines 121 and 122 store the current time so that it can be used as a unique filename. Line 134 writes the drain voltage test parameters to the top of the text document. Line 138 writes the gate voltage test parameters to the text document if the device is a HEMT. The for-loop in lines 145 through 149 is where the script is actually writing 3 columns to the text file, one column for a data reference number, one column for voltage, and one column for current. (See example results text file in Section 4.4.) Results text files are saved to the same directory the script is located.

Note: The specific way these text files are written was purely based on the way we wrote our data analysis scripts. These scripts can be changed and modified to accommodate the format requirements for each individual user.

3.1.2 Run the I_{ds}–V_{gs} Test

The script performs an I_{ds}–V_{gs} measurement on a HEMT (cannot be performed on a diode). The logic and setup for this task is very similar to task 1, the only difference is the actual 4155C variable setup, which would change the I-V measurement to an I_{ds}–V_{gs} measurement. Between lines 163 through 173, the script is checking for the specific device type and if test parameters have been set. In lines 177 through 196, the script is telling the 4155C to set up its variables to perform an I_{ds}–V_{gs} measurement. These specific commands are chosen based on how a user would manually set up an I_{ds}–V_{gs} measurement on the front panel.[1] In lines 198 through 206, the script is setting test parameter values for each variable based on the stored values in the script (these said values are stored using command 6). Lines 208 through 211 are telling the script to format the data displayed onto the front panel of the 4155C.

In the next section, line 217 to line 250 the script is formatting the output data so that they can be written to a text file in a certain way so that data analysis/parsing can be easily performed. This section is identical to its counterpart in task 1. Lines 222 and 224 are the lines telling the 4155C to store the ASCII data for I_D and V_G (note the usage of V_G and not V_D). See task 1 for an explanation on the text file writing and formatting. Results text files will be saved to the same directory the script is in.

3.1.3 Run the OFF-State Breakdown Voltage Test

In this task, the script performs an OFF-state breakdown voltage measurement on a HEMT. (Note: The measurement cannot be performed on a diode.) The logic and setup for this task is similar to tasks 1 and 2; the main difference is the actual 4155C variable setup. Between lines 256 and 266, the script is checking for the specific device type and if test parameters have been set. In lines 270 through 289, the script is telling the 4155C to set up its variables to perform an OFF-state breakdown voltage measurement. These specific commands are selected based on how a user would manually set up the measurement on the front panel.[1] In lines 291 through 300, the script is setting test parameter values for each variable based on the stored values in the script (those said values are stored using command 7).

In the next section, between lines 308 and 341, the script is formatting the output data so that it can be written to a text file in a specific way so that data analysis/parsing can be performed. This section is identical to its counterpart in tasks 1 and 2. Lines 313 and 315 tell the 4155C to store the ASCII data for I_D and V_D. (See task 1 for an explanation on the text file writing and formatting.) Results text files will be saved to the same directory the script is in.

3.1.4 Choose the Device Type ("hemt" or "diode")

Line 347, `device_type = raw_input("Enter the device type ('hemt' or 'diode'): ")`, is assigning a user-input device type to the variable `device_type`. This variable is then used through the script to keep track of what measurements can and cannot be performed for the given device (e.g., I_{ds}–V_{gs} cannot be performed on a diode). Line 349 through 357 is checking to see if the input is entered correctly as either `"hemt"` or `"diode"`.

3.1.5 Set the I-V Test Parameters

In this section, the script prompts the user to enter a set of test parameters that will be used to run command 1, the I-V test. These parameters are selected based on which parameters are needed to execute an I-V test.[1] Between lines 365 and 389, 2 cases are made, one for HEMT devices and one for diode devices. Based on which device type is currently selected, the user prompt for test parameter input will change. For HEMT I-V tests, the drain and gate parameters both need to be set, whereas for diode devices, only one variable needs to be set.

3.1.6 Set the I_{ds}–V_{gs} Test Parameters

In this task, the script prompts the user to enter a set of test parameters that will be used to run command 2, the I_{ds}–V_{gs} test. These parameters are selected based on which parameters are needed to execute an I_{ds}–V_{gs} test.[1] In comparison to the I-V

test, the I_{ds}–V_{gs} is only performed on HEMT devices. Lines 401 through 414 show where the script prompts the user for inputs.

3.1.7 Set the OFF-State Breakdown Voltage Test Parameters

In this task, the script prompts the user to enter a set of test parameters that will be used to run command 3, the OFF-state breakdown voltage test. These parameters are selected based on which parameters are needed to execute the test.[1] The OFF-state breakdown voltage test is only performed on HEMT devices. Lines 426 through 435 show where the script prompts the user for inputs. Specifically, line 433, `const_gate_voltage = raw_input("Enter Constant Gate Voltage (V): ")`, is prompting the user for a gate voltage that is in the OFF state of the device, below the threshold voltage.

3.1.8 Display User Input Test Parameters

The script displays all the test parameters for each of the 3 measurement tests. Lines 447 through 484 encompass the HEMT test parameters; each if-else statement is simply checking to see if any of the said test parameters have been entered, else display nothing. Lines 490 through 501 encompass the diode test parameters; the if-else statement is simply checking to see if the test parameters have been entered, else display nothing.

3.1.9 Exit Program

The script exits the entire program. Line 505, `sys.exit()`, uses the sys package that comes with Python to exit system processes.

3.2 Micromanipulator P200L Semiautomatic Probing Station

In this section, we discuss the scripts that integrate the P200L semiautomatic probing station with the 4155C semiconductor parameter analyzer. In comparison to the manual setup with a new UI, this script does not have a UI and is essentially an automated testing program.

Script 2 (Prober_HEMT_withCheck_IdsVds.py) is shown in detail in Section 4.2. It is important to note that the scripts shown in this technical report are written to take I-V measurements for diodes and I_{ds}–V_{ds} measurements for HEMTs. These scripts can easily be changed to take I_{ds}–V_{gs} and OFF-state breakdown voltage measurements, as well; however, in comparison to script 1, the source code needs to change rather than simply using a UI. Future technical reports on this topic will include these scripts once they have been prepared.

Scripts 2 and 3 are nearly identical in their logic and organization. The only difference is that script 2 is for HEMTs I_{ds}–V_{ds} measurements and script 3 is for diode I-V measurements. Information shown in this section refers to script 2; this information can also be applied to script 3 (see Section 4.3).

Note: The P200L automated Python scripts written for this technical report assumes from the start that the probe tips are making contact with the upper-leftmost device in the sample.

Identical to the script 1, lines 13 and 14 establish connection with the 4155C. Line 16 eliminates the default 5-s timeout and allows the program to execute its commands without time restriction (see Section 2). In lines 18 and 19, we are importing the NewportESP301/NewportESP301Axis directory from the InstrumentKits Python package.[2] This Python package is specifically built to control the ESP301 motion controller, which moves the P200L stage. In line 21, `controller = NewportESP301.open_serial("COM4", 921600)`, the script is directly establishing connection with the P200L, COM4 refers to the communication port, and 921600 represents the baud rate of that connection.[3]

In lines 24 through 42, the script is setting the axis configuration. Lines 28 through 42 are setting the acceleration, deceleration, velocity, and max velocity of the 3 axes (the axes are represented by the second argument) to a value so that motion jerking and vibrations are under control on the plate. Acceleration/deceleration are in terms of millimeters per seconds squared and both velocity and max velocity are in terms of millimeters per second. These values are subject to adjustment depending on device type, size, and quantity.

In the next section, the quantity, spacing, and size of devices are used to establish correct testing patterns. Lines 44 and 45 prompt the user to input the number of rows and columns of devices for the sample. Line 48, `space_between_columns = numpy.zeros((num_of_rows, devices_LtR-1))`, creates a 2-D array that represents the space between each column (device) from left to right. Lines 57 through 59 now fill up this 2-D array with the spacing values. This section of the source code needs to be coded directly to adjust the spacing or number of rows/columns. For example if there is another row of devices, simply add `space_between_columns [3] = […,…,…,…,…]`. This logic is oriented very intuitively so that different users with different requirements can easily modify the source code. It is recommended that this script be modified for different device layouts or mask sets. Thus, each mask set will have a different script to accommodate the geometry of said mask set.

In the next section, lines 69 through 102 perform a row and column alignment check. This section also shows how to move the ESP301 axes. See lines 70 and 71,

`NewportESP301Axis(controller,2).move(-0.08, absolute=False)` and `NewportESP301Axis(controller,2). Wait_for_stop()`, respectively. Line 70 is essentially telling Axis 3 (2 in the code since it starts counting from 0) to move the plate down 0.08 mm (see Section 1 for more details on axis direction). Line 71 tells the program to wait for the plate to finish moving before going onto the next command.

In terms of the specific alignment checking logic, an example process is as follows:

1) Move from the first device's position in the row to the "intended" last device position in the row. Prompt the user to check microscope to see if the device in the row is indeed making contact with the probe. If the probe is making contact, proceed, else, move probe back to initial position and prompt the user to attempt manual alignment with chuck rotation. See lines 69 through 84.

Assuming the row was aligned correctly, the next step would be as follows:

2) Move from the first device's position in the column to the "intended" last device position in the column. Prompt the user to check microscope to see if the last device in the column is indeed making contact with the probe. If the probe is making contact, proceed, else, move probe back to initial position and prompt the user to attempt manual alignment with chuck rotation. See lines 87 through 102.

Once this alignment checking process is complete and the alignment of the sample has been verified, the script performs an I-V test on the first device in the top-left corner and then prompts the user to tell the program if the data taken are viable. This process was implemented to check if the probes are making good contact with the devices at the given height (e.g., if data are not being taken correctly then the probe tip may be too high up and not making good contact). See lines 106 through 145.

In the following section, once the script has confirmed that the rows and columns as well as probe contact are aligned properly, it will begin its automated device testing and data collection process. In lines 151 and 153, notice that 2 nested for-loops are used. These for-loops regulate the entire automation process. The outer for-loop cycles through each row, once all the devices in the row have been measured it moves on to the next row. The inner for-loop cycles through each column or, in other words, each device in the current row. Moving into the nested loops, lines 158 through 230 are the data collection portion of this automated program. Notice that this entire portion of the code is nearly identical to the I-V/ I_{ds}–V_{ds} measurement that was shown in script 1, because we automated that exact

same I-V/ I_{ds}–V_{ds} data collection process to run within a loop for HEMTs specifically (see Section 3, task 1 HEMTs for details on how this segment of code works as a whole). By using the exact same process, script 2's automated I_{ds}–V_{ds} tests on HEMTs can be adapted to perform both I_{ds}–V_{gs} and OFF-state breakdown voltage tests by simply implementing code into script 2 from script 1, tasks 2 and 3. The only major difference that should be noticed is that script 2 has most of the variables and write statements hard coded. This is because, as mentioned previously, there is no UI for the semiautomatic probing script so these things need to be changed at the source code level. It is also important to explain that if test parameters for this test need to be changed, the source code, lines 127 through 135, lines 181 through 189, and lines 209 through 214 need to be modified accordingly.

In the last portion of this script, lines 235 through 265 control the actual automated movement of the probing station. Notice that this segment of code is filled with axes movement commands similar to lines 70 and 71, which were explained previously. Line 235, `controller.wait(2500)`, is telling the prober to wait 2.5 s just to make sure the data collection process is done before moving to the next device. The process and pattern of the prober movement is as follows.

Starting in the inner for-loop, this portion controls the prober movement between each column (device):

1) Move the plate down 0.08 mm so that the probe tips are no longer making contact with the sample. See lines 237 and 238.

2) If the current device is not the last device in the row, move right however much depending on device spacing between each column (stored in the original 2-D array grid in lines 57 through 59). If the current device is the last device in the row, there will be no movement further right. See lines 240 through 242.

3) Move the plate back up 0.08 mm so that the probe tips are making contact with the sample. See lines 244 and 245.

Once the last device in the row has been measured, the script will then proceed into the outer for-loop where it controls movement between each row. Depending on whether it is the last row in the sample, the script will reenter the inner for-loop and proceed to take data across the row again. For the outer for-loop, do the following:

1) Move the plate down 0.08 mm so that the probe tips are no longer making contact with the sample. See lines 247 and 248.

2) Move the plate back to the first device in its current row, hence, `sum(space_between_columns [row])` in line 250, which sums up the

spacing of all the devices in that row. This means that the prober will start on the leftmost device in the next row. See lines 250 and 251.

3) If the current row is not the last row, move toward the user (to the next row) however much depending on device spacing between each row (stored in the 1-D array in line 62). If the current row is the last row in the sample, move back to the very first row in the sample since this means that testing has been completed. See lines 253 through 262.

4) Move the plate back up 0.08 mm so that the probe tips are making contact with the sample. See lines 264 and 265.

Note: Scripts 2 and 3 save the generated results files to the current directory that the script is in. The text file saving process and formatting process is identical to script 1 except that certain parameters are hard coded into scripts 2 and 3 (see script 1, task 1 for explanation on formatting portion of the code).

4. Scripts and Text Files

4.1 Script 1: HEMT_Diode_toComputer_withInterface_All3.py

Script 1 is used for manual probing with the 4155C. The program is used to run I-V, I_{ds}–V_{gs}, and OFF-state breakdown voltage tests with the 4155C and manual probe station. This program automatically saves each test run data as a file onto the computer. The file is named after the time the measurement was taken in HHMMSS format. Results files are saved to the same directory this script is located.

Note: This program is *not* created to run with the P200L semiautomatic probing station.

Script Location: L:\WBG Team\python_scripts\...
Language: Python, version 2.7
Script Version: 1.0
Date: 08/08/2016
POC: Randy Tompkins (randy.tompkins.civ@mail.mil)

```
from datetime import datetime
import visa
import sys

def intorfloat(s):
    f = float(s)
    i = int(f)
    return i if i == f else f

rm = visa.ResourceManager()
instr = rm.open_resource("GPIB0::17::INSTR")

del instr.timeout

device_type = "None"

choice5_picked_once_hemt = 0
choice5_picked_once_diode = 0
choice6_picked_once = 0
choice7_picked_once = 0

while True:

    print ("\n##################################################################\n")
    print ("Current Selected Device Type: %s\n" % device_type)
    print ("Please enter 1, 2, 3, 4, 5, 6, 7, 8, or 9 to choose from an option below.\n")
    print ("1. RUN **IV** Test\n")
    print ("2. RUN **Ids - Vgs** Test\n")
    print ("3. RUN OFF-State Breakdown Voltage Test\n")
    print ("4. Choose Device Type ('hemt' or 'diode')\n")
    print ("5. SET **IV** Test Parameters\n")
    print ("6. SET **Ids - Vgs** Test Parameters\n")
    print ("7. SET OFF-State Breakdown Voltage Test Parameters\n")
    print ("8. Display User Input Test Parameters\n")
    print ("9. Exit Program")
    print ("\n##################################################################")

    choice = raw_input(":")

    if choice == "1":

        if device_type == "None":

            print ("\nERROR: You have not chosen a device type yet.")

```

```
        elif (device_type == "hemt") and (choice5_picked_once_hemt == 0):

            print ("\nERROR: You have not initialized IV test parameters for a HEMT.")

        elif (device_type == "diode") and (choice5_picked_once_diode == 0):

            print ("\nERROR: You have not initialized IV test parameters for a DIODE.")

        else:

            instr.write("*RST")

            if device_type == "hemt":

                instr.write(":PAGE:CHAN:MODE SWEEP")
                instr.write(":PAGE:CHAN:SMU4:DIS")
                instr.write(":PAGE:CHAN:VSU1:DIS")
                instr.write(":PAGE:CHAN:VSU2:DIS")
                instr.write(":PAGE:CHAN:VMU1:DIS")
                instr.write(":PAGE:CHAN:VMU2:DIS")
                instr.write(":PAGE:CHAN:SMU1:VNAME 'VD' ")
                instr.write(":PAGE:CHAN:SMU2:VNAME 'VS' ")
                instr.write(":PAGE:CHAN:SMU3:VNAME 'VG' ")
                instr.write(":PAGE:CHAN:SMU1:INAME 'ID' ")
                instr.write(":PAGE:CHAN:SMU2:INAME 'IS' ")
                instr.write(":PAGE:CHAN:SMU3:INAME 'IG' ")
                instr.write(":PAGE:CHAN:SMU1:MODE V")
                instr.write(":PAGE:CHAN:SMU1:FUNC VAR1")
                instr.write(":PAGE:CHAN:SMU2:MODE COMM")
                instr.write(":PAGE:CHAN:SMU2:FUNC CONS")
                instr.write(":PAGE:CHAN:SMU3:MODE V")
                instr.write(":PAGE:CHAN:SMU3:FUNC VAR2")

            elif device_type == "diode":

                instr.write(":PAGE:CHAN:MODE SWEEP")
                instr.write(":PAGE:CHAN:SMU3:DIS")
                instr.write(":PAGE:CHAN:SMU4:DIS")
                instr.write(":PAGE:CHAN:VSU1:DIS")
                instr.write(":PAGE:CHAN:VSU2:DIS")
                instr.write(":PAGE:CHAN:VMU1:DIS")
                instr.write(":PAGE:CHAN:VMU2:DIS")
                instr.write(":PAGE:CHAN:SMU1:VNAME 'VD' ")
                instr.write(":PAGE:CHAN:SMU2:VNAME 'VS' ")
                instr.write(":PAGE:CHAN:SMU1:INAME 'ID' ")
```

```
91.                 instr.write(“:PAGE:CHAN:SMU2:INAME  ‘IS’ ”)
92.                 instr.write(“:PAGE:CHAN:SMU1:MODE V”)
93.                 instr.write(“:PAGE:CHAN:SMU1:FUNC VAR1”)
94.                 instr.write(“:PAGE:CHAN:SMU2:MODE COMM”)
95.                 instr.write(“:PAGE:CHAN:SMU2:FUNC CONS”)
96.
97.             instr.write(“:PAGE:MEAS:VAR1:START %f”  % float(start1))
98.             instr.write(“:PAGE:MEAS:VAR1:STOP %f”  % float(stop1))
99.             instr.write(“:PAGE:MEAS:VAR1:STEP %f”  % float(step1))
100.            instr.write(“:PAGE:MEAS:VAR1:COMP %f”  % float(comp1))
101.
102.            if device_type ==  “diode”:
103.
104.                instr.write(“:PAGE:MEAS:VAR1:POINTS %f”  % float(point1
    ))
105.
106.            if device_type ==  “hemt”:
107.
108.                instr.write(“:PAGE:MEAS:VAR2:START %f”  % float(start2)
    )
109.                instr.write(“:PAGE:MEAS:VAR2:STOP %f”  % float(stop2))

110.                instr.write(“:PAGE:MEAS:VAR2:STEP %f”  % float(step2))

111.                instr.write(“:PAGE:MEAS:VAR2:COMP %f”  % float(comp2))

112.                instr.write(“:PAGE:MEAS:VAR2:POINTS %f”  % float(point2
    ))
113.
114.            instr.write(“:PAGE:MEAS:SSTOP COMP”)
115.
116.            # Execute measurement. Single sweep.
117.            instr.write(“:PAGE:SCON:SING”)
118.            instr.write(“*WAI”)
119.
120.            # Record time of measurement.
121.            cur_time = str(datetime.now().time())
122.            formatted_cur_time = cur_time[0:2] + cur_time[3:5] + cur_tim
    e[6:8]
123.
124.            instr.write(“:FORM:DATA ASC”)
125.
126.            I_data = instr.query_ascii_values(“:DATA?  ‘ID’   “)
127.
128.            V_data = instr.query_ascii_values(“:DATA?  ‘VD’   “)
129.
130.            # Section details writing the new found data to a file on th
    e computer.
131.            file = open(“%s.txt”  % formatted_cur_time,  “w”)
```

```
132.
133.             file.write("\n")
134.             file.write("VD= %s to %s in %s step\n" % ( str(intorfloat(
    start1)),str(intorfloat(stop1)),str(intorfloat(step1)) ) )
135.
136.             if device_type == "hemt":
137.
138.                 file.write("VG= %s to %s in %s step\n" % ( str(intorfl
    oat(start2)),str(intorfloat(stop2)),str(intorfloat(step2)) ) )
139.
140.             file.write("NO. VD ID\n")
141.             file.write(" V A\n")
142.
143.             i = 1
144.
145.             for x, y in zip(V_data,I_data):
146.
147.                 file.write(str(i) + "  " + str(intorfloat(x)) + "  " +
     str(y) + "\n")
148.
149.                 i = i + 1
150.
151.             file.close()
152.
153.             cur_time = None
154.             formatted_cur_time = None
155.
156.             instr.write(":PAGE:GLIS")
157.             instr.write(":PAGE:GLIS:SCAL:AUTO ONCE")
158.
159.             print ("\nTest Completed. Please check 4155C front panel f
    or any errors.")
160.
161.     elif choice == "2":
162.
163.         if device_type == "None":
164.
165.             print ("\nERROR: You have not chosen a device type yet.")

166.
167.         elif (device_type == "diode"):
168.
169.             print ("\nERROR: Your current device type is DIODE. No Ids
     - Vgs measurement available.")
170.
171.         elif (device_type == "hemt") and (choice6_picked_once == 0):

172.
```

```
            print ("\nERROR: You have not initialized Ids - Vgs test parameters for a HEMT.")

        else:

            instr.write("*RST")

            instr.write(":PAGE:CHAN:MODE SWEEP")
            instr.write(":PAGE:CHAN:SMU4:DIS")
            instr.write(":PAGE:CHAN:VSU1:DIS")
            instr.write(":PAGE:CHAN:VSU2:DIS")
            instr.write(":PAGE:CHAN:VMU1:DIS")
            instr.write(":PAGE:CHAN:VMU2:DIS")
            instr.write(":PAGE:CHAN:SMU1:VNAME 'VD' ")
            instr.write(":PAGE:CHAN:SMU2:VNAME 'VS' ")
            instr.write(":PAGE:CHAN:SMU3:VNAME 'VG' ")
            instr.write(":PAGE:CHAN:SMU1:INAME 'ID' ")
            instr.write(":PAGE:CHAN:SMU2:INAME 'IS' ")
            instr.write(":PAGE:CHAN:SMU3:INAME 'IG' ")
            instr.write(":PAGE:CHAN:SMU1:MODE V")
            instr.write(":PAGE:CHAN:SMU1:FUNC VAR2")
            instr.write(":PAGE:CHAN:SMU2:MODE COMM")
            instr.write(":PAGE:CHAN:SMU2:FUNC CONS")
            instr.write(":PAGE:CHAN:SMU3:MODE V")
            instr.write(":PAGE:CHAN:SMU3:FUNC VAR1")

            instr.write(":PAGE:MEAS:VAR1:START %f" % float(g_start1))
            instr.write(":PAGE:MEAS:VAR1:STOP %f" % float(g_stop1))
            instr.write(":PAGE:MEAS:VAR1:STEP %f" % float(g_step1))
            instr.write(":PAGE:MEAS:VAR1:COMP %f" % float(g_comp1))
            instr.write(":PAGE:MEAS:VAR2:START %f" % float(g_start2))
            instr.write(":PAGE:MEAS:VAR2:STOP %f" % float(g_stop2))
            instr.write(":PAGE:MEAS:VAR2:STEP %f" % float(g_step2))
            instr.write(":PAGE:MEAS:VAR2:COMP %f" % float(g_comp2))
            instr.write(":PAGE:MEAS:VAR2:POINTS %f" % float(g_point2))

            instr.write(":PAGE:DISP:GRAP:Y1:NAME 'ID' ")
            instr.write(":PAGE:DISP:LIST:DEL 'VG' ")
            instr.write(":PAGE:DISP:LIST:DEL 'IG' ")
            instr.write(":PAGE:DISP:LIST 'VG', 'ID' ")

            instr.write(":PAGE:SCON:SING")
            instr.write("*WAI")

            # Record time of measurement.
            cur_time = str(datetime.now().time())
```

```
            formatted_cur_time = cur_time[0:2] + cur_time[3:5] + cur_time[6:8]

            instr.write(":FORM:DATA ASC")

            I_data = instr.query_ascii_values(":DATA? 'ID' ")

            V_data = instr.query_ascii_values(":DATA? 'VG' ")

            # Section details writing the new found data to a file on the computer.
            file = open("%s.txt" % formatted_cur_time, "w")

            file.write("\n")
            file.write("VD= %s to %s in %s step\n" % ( str(intorfloat(g_start2)),str(intorfloat(g_stop2)),str(intorfloat(g_step2)) ) )
            file.write("VG= %s to %s in %s step\n" % ( str(intorfloat(g_start1)),str(intorfloat(g_stop1)),str(intorfloat(g_step1)) ) )

            file.write("NO. VG ID\n")
            file.write(" V A\n")

            i = 1

            for x, y in zip(V_data,I_data):

                file.write(str(i) + " " + str(intorfloat(x)) + " " + str(y) + "\n")

                i = i + 1

            file.close()

            cur_time = None
            formatted_cur_time = None

            instr.write(":PAGE:GLIS")
            instr.write(":PAGE:GLIS:SCAL:AUTO ONCE")

            print ("\nTest Completed. Please check 4155C front panel for any errors.")

    elif choice == "3":

        if device_type == "None":

            print ("\nERROR: You have not chosen a device type yet.")

```

```
260.         elif (device_type ==  “diode” ):
261.
262.             print ( “\nERROR: Your current device type is DIODE. No OFF
    -State Breakdown Voltage measurement available.” )
263.
264.         elif (device_type ==  “hemt” ) and (choice7_picked_once == 0):

265.
266.             print ( “\nERROR: You have not initialized OFF-
    State Breakdown Voltage Test parameters for a HEMT.” )
267.
268.         else:
269.
270.             instr.write( “*RST” )
271.
272.             instr.write( “:PAGE:CHAN:MODE SWEEP” )
273.             instr.write( “:PAGE:CHAN:SMU4:DIS” )
274.             instr.write( “:PAGE:CHAN:VSU1:DIS” )
275.             instr.write( “:PAGE:CHAN:VSU2:DIS” )
276.             instr.write( “:PAGE:CHAN:VMU1:DIS” )
277.             instr.write( “:PAGE:CHAN:VMU2:DIS” )
278.             instr.write( “:PAGE:CHAN:SMU1:VNAME  ‘VD’ ” )
279.             instr.write( “:PAGE:CHAN:SMU2:VNAME  ‘VS’ ” )
280.             instr.write( “:PAGE:CHAN:SMU3:VNAME  ‘VG’ ” )
281.             instr.write( “:PAGE:CHAN:SMU1:INAME  ‘ID’ ” )
282.             instr.write( “:PAGE:CHAN:SMU2:INAME  ‘IS’ ” )
283.             instr.write( “:PAGE:CHAN:SMU3:INAME  ‘IG’ ” )
284.             instr.write( “:PAGE:CHAN:SMU1:MODE V” )
285.             instr.write( “:PAGE:CHAN:SMU1:FUNC VAR1” )
286.             instr.write( “:PAGE:CHAN:SMU2:MODE COMM” )
287.             instr.write( “:PAGE:CHAN:SMU2:FUNC CONS” )
288.             instr.write( “:PAGE:CHAN:SMU3:MODE V” )
289.             instr.write( “:PAGE:CHAN:SMU3:FUNC VAR2” )
290.
291.             instr.write( “:PAGE:MEAS:VAR1:START %f”  % float(vb_start1))

292.             instr.write( “:PAGE:MEAS:VAR1:STOP %f”  % float(vb_stop1))
293.             instr.write( “:PAGE:MEAS:VAR1:STEP %f”  % float(vb_step1))
294.             instr.write( “:PAGE:MEAS:VAR1:COMP %f”  % float(vb_comp1))
295.
296.             instr.write( “:PAGE:MEAS:VAR2:START %f”  % float(const_gate_
    voltage))
297.             instr.write( “:PAGE:MEAS:VAR2:STOP %f”  % float(const_gate_v
    oltage))
298.             instr.write( “:PAGE:MEAS:VAR2:STEP 1” )
299.             instr.write( “:PAGE:MEAS:VAR2:COMP 0.1” )
300.             instr.write( “:PAGE:MEAS:VAR2:POINTS 1” )
301.
302.             instr.write( “:PAGE:MEAS:SSTOP COMP” )
```

```
303.
304.             instr.write(":PAGE:SCON:SING")
305.             instr.write("*WAI")
306.
307.             # Record time of measurement.
308.             cur_time = str(datetime.now().time())
309.             formatted_cur_time = cur_time[0:2] + cur_time[3:5] + cur_tim
    e[6:8]
310.
311.             instr.write(":FORM:DATA ASC")
312.
313.             I_data = instr.query_ascii_values(":DATA?  'ID'   ")
314.
315.             V_data = instr.query_ascii_values(":DATA?  'VD'   ")
316.
317.             # Section details writing the new found data to a file on th
    e computer.
318.             file = open("%s.txt" % formatted_cur_time,  "w")
319.
320.             file.write("\n")
321.             file.write("VD= %s to %s in %s step\n" % ( str(intorfloat(
    vb_start1)),str(intorfloat(vb_stop1)),str(intorfloat(vb_step1)) ) )

322.             file.write("VG= %s to %s in 1 step\n" % ( str(intorfloat(c
    onst_gate_voltage)),str(intorfloat(const_gate_voltage)) ) )
323.
324.             file.write("NO.  VD  ID\n")
325.             file.write("  V  A\n")
326.
327.             i = 1
328.
329.             for x,  y in zip(V_data,I_data):
330.
331.                 file.write(str(i) +  "   " + str(intorfloat(x)) +  "   " +
     str(y) +  "\n")
332.
333.                 i = i + 1
334.
335.             file.close()
336.
337.             cur_time = None
338.             formatted_cur_time = None
339.
340.             instr.write(":PAGE:GLIS")
341.             instr.write(":PAGE:GLIS:SCAL:AUTO ONCE")
342.
343.             print ("\nTest Completed. Please check 4155C front panel f
    or any errors.")
344.
```

```
345.     elif choice == "4" :
346.
347.         device_type = raw_input( "Enter the device type ( 'hemt' or 'd
    iode' ): ")
348.
349.         if device_type == "hemt" or device_type == "diode" :
350.
351.             pass
352.
353.         else:
354.
355.             print ( "\nERROR: Invalid Device Type." )
356.
357.             device_type = "None"
358.
359.     elif choice == "5" :
360.
361.         if device_type == "None" :
362.
363.             print ( "\nERROR: You have not chosen a device type yet." )

364.
365.         elif device_type == "hemt" :
366.
367.             print ( "\nSet **IV** Parameters for HEMT:" )
368.             start1 = raw_input( "Enter variable 1 (Drain) START: ")
369.             stop1 = raw_input( "Enter variable 1 (Drain) STOP: ")
370.             step1 = raw_input( "Enter variable 1 (Drain) STEP: ")
371.             comp1 = raw_input( "Enter variable 1 (Drain) COMPLIANCE: ")

372.             start2 = raw_input( "Enter variable 2 (Gate) START: ")
373.             stop2 = raw_input( "Enter variable 2 (Gate) STOP: ")
374.             step2 = raw_input( "Enter variable 2 (Gate) STEP: ")
375.             comp2 = raw_input( "Enter variable 2 (Gate) COMPLIANCE: ")

376.             point2 = raw_input( "Enter variable 2 (Gate) POINTS: ")
377.
378.             choice5_picked_once_hemt = 1
379.
380.         elif device_type == "diode" :
381.
382.             print ( "\nSet **IV** Parameters for DIODE:" )
383.             start1 = raw_input( "Enter variable 1 (Drain) START: ")
384.             stop1 = raw_input( "Enter variable 1 (Drain) STOP: ")
385.             step1 = raw_input( "Enter variable 1 (Drain) STEP: ")
386.             comp1 = raw_input( "Enter variable 1 (Drain) COMPLIANCE: ")

387.             point1 = raw_input( "Enter variable 1 (Drain) POINTS: ")
388.
```

```
            choice5_picked_once_diode = 1

    elif choice == "6":

        if device_type == "None":

            print ("\nERROR: You have not chosen a device type yet.")

        elif device_type == "diode":

            print ("\nERROR: Your current device type is DIODE. No Ids - Vgs measurement available.")

        elif device_type == "hemt":

            print ("\nSet **Ids - Vgs** Parameters for HEMT:")
            g_start1 = raw_input("Enter variable 1 (Gate) START: ")
            g_stop1 = raw_input("Enter variable 1 (Gate) STOP: ")
            g_step1 = raw_input("Enter variable 1 (Gate) STEP: ")
            g_comp1 = raw_input("Enter variable 1 (Gate) COMPLIANCE: ")
            g_start2 = raw_input("Enter variable 2 (Drain) START: ")
            g_stop2 = raw_input("Enter variable 2 (Drain) STOP: ")
            g_step2 = raw_input("Enter variable 2 (Drain) STEP: ")
            g_comp2 = raw_input("Enter variable 2 (Drain) COMPLIANCE: ")
            g_point2 = raw_input("Enter variable 2 (Drain) POINTS: ")

            choice6_picked_once = 1

    elif choice == "7":

        if device_type == "None":

            print ("\nERROR: You have not chosen a device type yet.")

        elif device_type == "diode":

            print ("\nERROR: Your current device type is DIODE. No OFF-State Breakdown Voltage measurement available.")

        elif device_type == "hemt":

            print ("\nSet OFF-State Breakdown Voltage Test Parameters for HEMT:")
```

```
429.            vb_start1 = raw_input( “Enter variable 1 (Drain) START:  “)
430.            vb_stop1 = raw_input( “Enter variable 1 (Drain) STOP:  “)
431.            vb_step1 = raw_input( “Enter variable 1 (Drain) STEP:  “)
432.            vb_comp1 = raw_input( “Enter variable 1 (Drain) COMPLIANCE:
    “)
433.            const_gate_voltage = raw_input( “Enter Constant Gate Voltage
    (V):  “)
434.
435.            choice7_picked_once = 1
436.
437.    elif choice ==  “8” :
438.
439.        if device_type ==  “None” :
440.
441.            print ( “\nERROR: You have not chosen a device type yet.” )
442.
443.        elif device_type ==  “hemt” :
444.
445.            print ( “\nCurrent Selected Device Type: %s”  % device_type
   )
446.
447.            if choice5_picked_once_hemt == 1:
448.
449.                print ( “\n**IV** Parameters for HEMT:\n” )
450.                print ( “Variable 1 (Drain) START: %s”  % start1)
451.                print ( “Variable 1 (Drain) STOP: %s”  % stop1)
452.                print ( “Variable 1 (Drain) STEP: %s”  % step1)
453.                print ( “Variable 1 (Drain) COMPLIANCE: %s”  % comp1)
454.                print ( “Variable 2 (Gate) START: %s”  % start2)
455.                print ( “Variable 2 (Gate) STOP: %s”  % stop2)
456.                print ( “Variable 2 (Gate) STEP: %s”  % step2)
457.                print ( “Variable 2 (Gate) COMPLIANCE: %s”  % comp2)
458.                print ( “Variable 2 (Gate) POINTS: %s”  % point2)
459.
460.            if choice6_picked_once == 1:
461.
462.                print ( “\n**Ids  -  Vgs** Parameters for HEMT:\n” )
463.                print ( “Variable 1 (Gate) START: %s”  % g_start1)
464.                print ( “Variable 1 (Gate) STOP: %s”  % g_stop1)
465.                print ( “Variable 1 (Gate) STEP: %s”  % g_step1)
466.                print ( “Variable 1 (Gate) COMPLIANCE: %s”  % g_comp1)
467.                print ( “Variable 2 (Drain) START: %s”  % g_start2)
468.                print ( “Variable 2 (Drain) STOP: %s”  % g_stop2)
469.                print ( “Variable 2 (Drain) STEP: %s”  % g_step2)
470.                print ( “Variable 2 (Drain) COMPLIANCE: %s”  % g_comp2)
```

```
471.                print (“Variable 2 (Drain) POINTS: %s” % g_point2)
472.
473.            if choice7_picked_once == 1:
474.
475.                print (“\nOFF-
   State Breakdown Voltage Test Parameters for HEMT:\n”)
476.                print (“Variable 1 (Drain) START: %s” % vb_start1)
477.                print (“Variable 1 (Drain) STOP: %s” % vb_stop1)
478.                print (“Variable 1 (Drain) STEP: %s” % vb_step1)
479.                print (“Variable 1 (Drain) COMPLIANCE: %s” % vb_comp1
   )
480.                print (“Breakdown Voltage (V): %s” % const_gate_volta
   ge)
481.
482.            if (choice5_picked_once_hemt == 0) and (choice6_picked_once
    == 0) and (choice7_picked_once == 0):
483.
484.                print (“\nNothing to Display. Test parameters have not
    been 23sci23rt2323ed yet.”)
485.
486.        elif device_type == “diode”:
487.
488.            print (“\nCurrent Selected Device Type: %s” % device_type
   )
489.
490.            if choice5_picked_once_diode == 1:
491.
492.                print (“\n**IV** Parameters for Diode:\n”)
493.                print (“Variable 1 (Drain) START: %s” % start1)
494.                print (“Variable 1 (Drain) STOP: %s” % stop1)
495.                print (“Variable 1 (Drain) STEP: %s” % step1)
496.                print (“Variable 1 (Drain) COMPLIANCE: %s” % comp1)
497.                print (“Variable 1 (Drain) POINTS: %s” % point1)
498.
499.            else:
500.
501.                print (“\nNothing to Display. Test parameters have not
    been 23sci23rt2323ed yet.”)
502.
503.    elif choice == “9”:
504.
505.        sys.exit()
506.
507.    else:
508.
509.        print (“\nERROR: You have entered an invalid choice.”)
510.        print (“Please Try Again.”)
511.
```

4.2 Script 2: Prober_HEMT_withCheck_IdsVds.py

Script 2 is used for HEMT semiautomatic probing with the P200L. The program used to run I-V tests with the Agilent 4155C semiconductor parameter analyzer and Micromanipulator P200L semiautomatic probe station both located at ARL on HEMT devices. This program automatically saves each device test as a file onto the computer. Results files are saved to the same directory this script is located.

The file is named after the time the measurement was taken in HHMMSS format.

Script Location: L:\WBG Team\python_scripts\...
Language: Python, version 2.7
Script Version: 1.0
Date: 08/08/2016
POC: Randy Tompkins (randy.tompkins.civ@mail.mil)

```
1.  import numpy
2.  import visa
3.  import sys
4.  import time
5.
6.  from datetime import datetime
7.
8.  def intorfloat(s):
9.      f = float(s)
10.     i = int(f)
11.     return i if i == f else f
12.
13. rm = visa.ResourceManager()
14. instr = rm.open_resource("GPIB0::17::INSTR") # Connect to the 4155C
15.
16. del instr.timeout
17.
18. from instruments.newport import NewportESP301
19. from instruments.newport import NewportESP301Axis
20.
21. controller = NewportESP301.open_serial("COM4", 921600) # Connect to th
    e P200L
22.
23. #This section sets the axis configurations. Subject to adjustment depend
    ing on device size/type.
24. NewportESP301Axis(controller,0).trajectory=1
25. NewportESP301Axis(controller,1).trajectory=1
26. NewportESP301Axis(controller,2).trajectory=1
27.
28. NewportESP301Axis(controller,0).acceleration = 0.5
29. NewportESP301Axis(controller,1).acceleration = 0.5
30. NewportESP301Axis(controller,2).acceleration = 0.5
```

```
31.
32. NewportESP301Axis(controller,0).deceleration = 0.5
33. NewportESP301Axis(controller,1).deceleration = 0.5
34. NewportESP301Axis(controller,2).deceleration = 0.5
35.
36. NewportESP301Axis(controller,0).velocity = 1
37. NewportESP301Axis(controller,1).velocity = 1
38. NewportESP301Axis(controller,2).velocity = 0.1
39.
40. NewportESP301Axis(controller,0).max_velocity = 1.5
41. NewportESP301Axis(controller,1).max_velocity = 1.5
42. NewportESP301Axis(controller,2).max_velocity = 0.2
43.
44. num_of_rows = int(raw_input( "Enter the number of rows:  "))
45. devices_LtR = int(raw_input( "Enter the number of devices left to right
    (columns):  "))
46.
47. #This initializes the 2D array that represents the space_between_columns
     between each device (left to right).
48. space_between_columns = numpy.zeros((num_of_rows, devices_LtR-1))
49.
50. #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
51.
52. #Please note that this portion of the code will change based on the mask
     design and the spacing between the individual devices that will vary wi
    th each design.  We may also need to change the number of rows.
53.
54. #IMPORTANT NOTE: These values are in mm.
55.
56. #This is simply setting the values in the 2D array to the spacing betwee
    n each device in each corresponding row.
57. space_between_columns [0] = [0.150,0.150,0.175,0.200,0.225]
58. space_between_columns [1] = [0.250,0.250,0.275,0.300,0.325]
59. space_between_columns [2] = [0.350,0.370,0.380,0.400,0.430]
60.
61. #This array represents the space_between_columns between each row (top t
    o bottom).
62. gap_between_rows = [0.350,  0.400]
63.
64. #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
65.
66. gap_counter = 0
67.
68. # System attempts to check row alignment.
69. print ( "\nSystem is attempting to locate the last device in the first R
    OW." )
70. NewportESP301Axis(controller,2).move(-0.08,  absolute=False)
71. NewportESP301Axis(controller,2).wait_for_stop()
```

```
NewportESP301Axis(controller,0).move(sum(space_between_columns [0]), absolute=False)
NewportESP301Axis(controller,0).wait_for_stop()
NewportESP301Axis(controller,2).move(0.08,absolute=False)
NewportESP301Axis(controller,2).wait_for_stop()

correctly_aligned_row = raw_input("Is the prober probing the last device in the first ROW correctly? Type 'yes' or 'no' : ")

NewportESP301Axis(controller,2).move(-0.08, absolute=False)
NewportESP301Axis(controller,2).wait_for_stop()
NewportESP301Axis(controller,0).move(-sum(space_between_columns [0]), absolute=False)
NewportESP301Axis(controller,0).wait_for_stop()
NewportESP301Axis(controller,2).move(0.08,absolute=False)
NewportESP301Axis(controller,2).wait_for_stop()

# System attempts to check column alignment.
print ("\nSystem is attempting to locate the last device in the first COLUMN.")
NewportESP301Axis(controller,2).move(-0.08, absolute=False)
NewportESP301Axis(controller,2).wait_for_stop()
NewportESP301Axis(controller,1).move(-sum(gap_between_rows), absolute=False)
NewportESP301Axis(controller,1).wait_for_stop()
NewportESP301Axis(controller,2).move(0.08,absolute=False)
NewportESP301Axis(controller,2).wait_for_stop()

correctly_aligned_column = raw_input("Is the prober probing the last device in the first COLUMN correctly? Type 'yes' or 'no' : ")

NewportESP301Axis(controller,2).move(-0.08, absolute=False)
NewportESP301Axis(controller,2).wait_for_stop()
NewportESP301Axis(controller,1).move(sum(gap_between_rows), absolute=False)
NewportESP301Axis(controller,1).wait_for_stop()
NewportESP301Axis(controller,2).move(0.08,absolute=False)
NewportESP301Axis(controller,2).wait_for_stop()

if correctly_aligned_row == "yes" and correctly_aligned_column == "yes":

    instr.write("*RST")

    instr.write(":PAGE:CHAN:MODE SWEEP")
    instr.write(":PAGE:CHAN:SMU4:DIS")
    instr.write(":PAGE:CHAN:VSU1:DIS")
    instr.write(":PAGE:CHAN:VSU2:DIS")
    instr.write(":PAGE:CHAN:VMU1:DIS")
```

```
    instr.write(“:PAGE:CHAN:VMU2:DIS”)
    instr.write(“:PAGE:CHAN:SMU1:VNAME  ‘VD’ ”)
    instr.write(“:PAGE:CHAN:SMU2:VNAME  ‘VS’ ”)
    instr.write(“:PAGE:CHAN:SMU3:VNAME  ‘VG’ ”)
    instr.write(“:PAGE:CHAN:SMU1:INAME  ‘ID’ ”)
    instr.write(“:PAGE:CHAN:SMU2:INAME  ‘IS’ ”)
    instr.write(“:PAGE:CHAN:SMU3:INAME  ‘IG’ ”)
    instr.write(“:PAGE:CHAN:SMU1:MODE V”)
    instr.write(“:PAGE:CHAN:SMU1:FUNC VAR1”)
    instr.write(“:PAGE:CHAN:SMU2:MODE COMM”)
    instr.write(“:PAGE:CHAN:SMU2:FUNC CONS”)
    instr.write(“:PAGE:CHAN:SMU3:MODE V”)
    instr.write(“:PAGE:CHAN:SMU3:FUNC VAR2”)

    instr.write(“:PAGE:MEAS:VAR1:START 0”)
    instr.write(“:PAGE:MEAS:VAR1:STOP 10”)
    instr.write(“:PAGE:MEAS:VAR1:STEP 0.1”)
    instr.write(“:PAGE:MEAS:VAR1:COMP 0.1”)
    instr.write(“:PAGE:MEAS:VAR2:START -5”  )
    instr.write(“:PAGE:MEAS:VAR2:STOP 1”)
    instr.write(“:PAGE:MEAS:VAR2:STEP 1”)
    instr.write(“:PAGE:MEAS:VAR2:COMP 0.01”)
    instr.write(“:PAGE:MEAS:VAR2:POINTS 7”)

    instr.write(“:PAGE:MEAS:SSTOP COMP”)

    instr.write(“:PAGE:SCON:SING”)
    instr.write(“*WAI”)

    instr.write(“:PAGE:GLIS”)
    instr.write(“:PAGE:GLIS:SCAL:AUTO ONCE”)

    correctly_probing = raw_input(“Is the prober taking correct data? T
ype  ‘yes’  or  ‘no’  :  “)

    if correctly_probing ==  “yes” :

        print  (“\nSystem will now begin probing the sample.”)

        for row in range(0, num_of_rows):

            for col in range(0, devices_LtR):

                #*****************************************************
**************************************************#
                #****************************************Data Collectio
n Section*****************************************#

                time.sleep(2)
```

```
159.
160.                 instr.write("*RST")
161.
162.                 instr.write(":PAGE:CHAN:MODE SWEEP")
163.                 instr.write(":PAGE:CHAN:SMU4:DIS")
164.                 instr.write(":PAGE:CHAN:VSU1:DIS")
165.                 instr.write(":PAGE:CHAN:VSU2:DIS")
166.                 instr.write(":PAGE:CHAN:VMU1:DIS")
167.                 instr.write(":PAGE:CHAN:VMU2:DIS")
168.                 instr.write(":PAGE:CHAN:SMU1:VNAME  'VD' ")
169.                 instr.write(":PAGE:CHAN:SMU2:VNAME  'VS' ")
170.                 instr.write(":PAGE:CHAN:SMU3:VNAME  'VG' ")
171.                 instr.write(":PAGE:CHAN:SMU1:INAME  'ID' ")
172.                 instr.write(":PAGE:CHAN:SMU2:INAME  'IS' ")
173.                 instr.write(":PAGE:CHAN:SMU3:INAME  'IG' ")
174.                 instr.write(":PAGE:CHAN:SMU1:MODE V")
175.                 instr.write(":PAGE:CHAN:SMU1:FUNC VAR1")
176.                 instr.write(":PAGE:CHAN:SMU2:MODE COMM")
177.                 instr.write(":PAGE:CHAN:SMU2:FUNC CONS")
178.                 instr.write(":PAGE:CHAN:SMU3:MODE V")
179.                 instr.write(":PAGE:CHAN:SMU3:FUNC VAR2")
180.
181.                 instr.write(":PAGE:MEAS:VAR1:START 0")
182.                 instr.write(":PAGE:MEAS:VAR1:STOP 10")
183.                 instr.write(":PAGE:MEAS:VAR1:STEP 0.1")
184.                 instr.write(":PAGE:MEAS:VAR1:COMP 0.1")
185.                 instr.write(":PAGE:MEAS:VAR2:START -5"  )
186.                 instr.write(":PAGE:MEAS:VAR2:STOP 1")
187.                 instr.write(":PAGE:MEAS:VAR2:STEP 1")
188.                 instr.write(":PAGE:MEAS:VAR2:COMP 0.01")
189.                 instr.write(":PAGE:MEAS:VAR2:POINTS 7")
190.
191.                 instr.write(":PAGE:MEAS:SSTOP COMP")
192.
193.                 instr.write(":PAGE:SCON:SING")
194.                 instr.write("*WAI")
195.
196.                 # Record time of measurement.
197.                 cur_time = str(datetime.now().time())
198.                 formatted_cur_time = cur_time[0:2] + cur_time[3:5] + cur
    _time[6:8]
199.
200.                 instr.write(":FORM:DATA ASC")
201.
202.                 I_data = instr.query_ascii_values(":DATA?  'ID'   ")
203.
204.                 V_data = instr.query_ascii_values(":DATA?  'VD'   ")
205.
```

```
                # Section details writing the new found data to a file on the computer.
                file = open("%s.txt" % formatted_cur_time, "w")

                file.write("\n")
                file.write("VD= 0 to 10 in 0.1 step\n")
                file.write("VG= -5 to 1 in 1 step\n")

                file.write("NO. VD ID\n")
                file.write(" V A\n")

                i = 1

                for x, y in zip(V_data,I_data):

                    file.write(str(i) + "  " + str(intorfloat(x)) + "  " + str(y) + "\n")

                    i = i + 1

                file.close()

                cur_time = None
                formatted_cur_time = None

                instr.write(":PAGE:GLIS")
                instr.write(":PAGE:GLIS:SCAL:AUTO ONCE")

                #*********************************************************************************************#
                #*********************************************************************************************#

                controller.wait(2500) # Wait 2.5 seconds before moving just to make sure the device is finished collecting data.

                NewportESP301Axis(controller,2).move(-0.08, absolute=False)
                NewportESP301Axis(controller,2).wait_for_stop()

                if col != devices_LtR-1:
                    NewportESP301Axis(controller,0).move(float(space_between_columns [row] [col]), absolute=False)
                    NewportESP301Axis(controller,0).wait_for_stop()

                NewportESP301Axis(controller,2).move(0.08,absolute=False)
                NewportESP301Axis(controller,2).wait_for_stop()

```

```
247.             NewportESP301Axis(controller,2).move(-
    0.08, absolute=False)
248.             NewportESP301Axis(controller,2).wait_for_stop()
249.
250.             NewportESP301Axis(controller,0).move(-
    sum(space_between_columns [row]), absolute=False)
251.             NewportESP301Axis(controller,0).wait_for_stop()
252.
253.             if row != num_of_rows-1:
254.
255.                 NewportESP301Axis(controller,1).move(-
    gap_between_rows[gap_counter], absolute=False)
256.                 NewportESP301Axis(controller,1).wait_for_stop()
257.                 gap_counter = gap_counter + 1
258.
259.             elif row == num_of_rows-1:
260.
261.                 NewportESP301Axis(controller,1).move(sum(gap_between_row
    s), absolute=False)
262.                 NewportESP301Axis(controller,1).wait_for_stop()
263.
264.             NewportESP301Axis(controller,2).move(0.08,absolute=False)
265.             NewportESP301Axis(controller,2).wait_for_stop()
266.
267.     else:
268.
269.         print ("PROBE CONTACT ERROR: Please attempt to make proper con
    tact with the device and try again.")
270.
271.         sys.exit()
272.
273. else:
274.
275.     print ("ALIGNMENT ERROR: Please attempt to align the sample proper
    ly by hand by adjusting the sample position and run the script again.")

276.
277.     sys.exit()
278.
279. print ("\nRun Completed.")
```

4.3 Script 3: Prober_Diode_withCheck_IdsVds.py

Script 3 is used for diode semiautomatic probing with the P200L. The program used to run I-V tests with the Agilent 4155C semiconductor parameter analyzer and Micromanipulator P200L semiautomatic probe station both located at ARL on diode devices. This program automatically saves each device test as a file onto the computer. Results files are saved to the same directory this script is located.

The file is named after the time the measurement was taken in HHMMSS format.

Script Location: L:\WBG Team\python_scripts\...
Language: Python, version 2.7
Script Version: 1.0
Date: 08/08/2016
POC: Randy Tompkins (randy.tompkins.civ@mail.mil)

```
import numpy
import visa
import sys
import time

from datetime import datetime

def intorfloat(s):
    f = float(s)
    i = int(f)
    return i if i == f else f

rm = visa.ResourceManager()
instr = rm.open_resource("GPIB0::17::INSTR") # Connect to the 4155C

del instr.timeout

from instruments.newport import NewportESP301
from instruments.newport import NewportESP301Axis

controller = NewportESP301.open_serial("COM4", 921600) # Connect to the P200L

#This section sets the axis configurations. Subject to adjustment depending on device size/type.
NewportESP301Axis(controller,0).trajectory=1
NewportESP301Axis(controller,1).trajectory=1
NewportESP301Axis(controller,2).trajectory=1

NewportESP301Axis(controller,0).acceleration = 0.5
NewportESP301Axis(controller,1).acceleration = 0.5
NewportESP301Axis(controller,2).acceleration = 0.5

NewportESP301Axis(controller,0).deceleration = 0.5
NewportESP301Axis(controller,1).deceleration = 0.5
NewportESP301Axis(controller,2).deceleration = 0.5

NewportESP301Axis(controller,0).velocity = 1
NewportESP301Axis(controller,1).velocity = 1
NewportESP301Axis(controller,2).velocity = 0.1
```

```
39.
40. NewportESP301Axis(controller,0).max_velocity = 1.5
41. NewportESP301Axis(controller,1).max_velocity = 1.5
42. NewportESP301Axis(controller,2).max_velocity = 0.2
43.
44. num_of_rows = int(raw_input( “Enter the number of rows: “))
45. devices_LtR = int(raw_input( “Enter the number of devices left to right
    (columns): “))
46.
47. #This initializes the 2D array that represents the space_between_columns
     between each device (left to right).
48. space_between_columns = numpy.zeros((num_of_rows, devices_LtR-1))
49.
50. #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
51.
52. #Please note that this portion of the code will change based on the mask
     design and the spacing between the individual devices that will vary wi
    th each design. We may also need to change the number of rows.
53.
54. #IMPORTANT NOTE: These values are in mm.
55.
56. #This is simply setting the values in the 2D array to the spacing betwee
    n each device in each corresponding row.
57. space_between_columns [0] = [0.150,0.150,0.175,0.200,0.225]
58. space_between_columns [1] = [0.250,0.250,0.275,0.300,0.325]
59. space_between_columns [2] = [0.350,0.370,0.380,0.400,0.430]
60.
61. #This array represents the space_between_columns between each row (top t
    o bottom).
62. gap_between_rows = [0.350, 0.400]
63.
64. #~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~#
65.
66. gap_counter = 0
67.
68. # System attempts to check row alignment.
69. print ( “\nSystem is attempting to locate the last device in the first R
    OW.” )
70. NewportESP301Axis(controller,2).move(-0.08, absolute=False)
71. NewportESP301Axis(controller,2).wait_for_stop()
72. NewportESP301Axis(controller,0).move(sum(space_between_columns [0]), abs
    olute=False)
73. NewportESP301Axis(controller,0).wait_for_stop()
74. NewportESP301Axis(controller,2).move(0.08,absolute=False)
75. NewportESP301Axis(controller,2).wait_for_stop()
76.
77. correctly_aligned_row = raw_input( “Is the prober probing the last devic
    e in the first ROW correctly? Type ‘yes’ or ‘no’ : “)
78.
```

```
79. NewportESP301Axis(controller,2).move(-0.08, absolute=False)
80. NewportESP301Axis(controller,2).wait_for_stop()
81. NewportESP301Axis(controller,0).move(-
    sum(space_between_columns [0]), absolute=False)
82. NewportESP301Axis(controller,0).wait_for_stop()
83. NewportESP301Axis(controller,2).move(0.08,absolute=False)
84. NewportESP301Axis(controller,2).wait_for_stop()
85.
86. # System attempts to check column alignment.
87. print ("\nSystem is attempting to locate the last device in the first C
    OLUMN.")
88. NewportESP301Axis(controller,2).move(-0.08, absolute=False)
89. NewportESP301Axis(controller,2).wait_for_stop()
90. NewportESP301Axis(controller,1).move(-
    sum(gap_between_rows), absolute=False)
91. NewportESP301Axis(controller,1).wait_for_stop()
92. NewportESP301Axis(controller,2).move(0.08,absolute=False)
93. NewportESP301Axis(controller,2).wait_for_stop()
94.
95. correctly_aligned_column = raw_input("Is the prober probing the last de
    vice in the first COLUMN correctly? Type 'yes' or 'no' : ")
96.
97. NewportESP301Axis(controller,2).move(-0.08, absolute=False)
98. NewportESP301Axis(controller,2).wait_for_stop()
99. NewportESP301Axis(controller,1).move(sum(gap_between_rows), absolute=Fal
    se)
100. NewportESP301Axis(controller,1).wait_for_stop()
101. NewportESP301Axis(controller,2).move(0.08,absolute=False)
102. NewportESP301Axis(controller,2).wait_for_stop()
103.
104. if correctly_aligned_row == "yes" and correctly_aligned_column == "y
    es":
105.
106.     instr.write("*RST")
107.
108.     instr.write(":PAGE:CHAN:MODE SWEEP")
109.     instr.write(":PAGE:CHAN:SMU3:DIS")
110.     instr.write(":PAGE:CHAN:SMU4:DIS")
111.     instr.write(":PAGE:CHAN:VSU1:DIS")
112.     instr.write(":PAGE:CHAN:VSU2:DIS")
113.     instr.write(":PAGE:CHAN:VMU1:DIS")
114.     instr.write(":PAGE:CHAN:VMU2:DIS")
115.     instr.write(":PAGE:CHAN:SMU1:VNAME 'VD' ")
116.     instr.write(":PAGE:CHAN:SMU2:VNAME 'VS' ")
117.     instr.write(":PAGE:CHAN:SMU1:INAME 'ID' ")
118.     instr.write(":PAGE:CHAN:SMU2:INAME 'IS' ")
119.     instr.write(":PAGE:CHAN:SMU1:MODE V")
120.     instr.write(":PAGE:CHAN:SMU1:FUNC VAR1")
121.     instr.write(":PAGE:CHAN:SMU2:MODE COMM")
```

```
    instr.write(":PAGE:CHAN:SMU2:FUNC CONS")

    instr.write(":PAGE:MEAS:VAR1:START 0")
    instr.write(":PAGE:MEAS:VAR1:STOP 5")
    instr.write(":PAGE:MEAS:VAR1:STEP 0.01")
    instr.write(":PAGE:MEAS:VAR1:COMP 0.01")
    instr.write(":PAGE:MEAS:VAR1:POINTS 7")

    instr.write(":PAGE:MEAS:SSTOP COMP")

    instr.write(":PAGE:SCON:SING")
    instr.write("*WAI")

    instr.write(":PAGE:GLIS")
    instr.write(":PAGE:GLIS:SCAL:AUTO ONCE")

    correctly_probing = raw_input("Is the prober taking correct data? Type 'yes' or 'no' : ")

    if correctly_probing == "yes":

        print ("\nSystem will now begin probing the sample.")

        for row in range(0, num_of_rows):

            for col in range(0, devices_LtR):

                #*********************************************************************************************#
                #*****************************************Data Collection Section*****************************#

                time.sleep(2)

                instr.write("*RST")

                instr.write(":PAGE:CHAN:MODE SWEEP")
                instr.write(":PAGE:CHAN:SMU3:DIS")
                instr.write(":PAGE:CHAN:SMU4:DIS")
                instr.write(":PAGE:CHAN:VSU1:DIS")
                instr.write(":PAGE:CHAN:VSU2:DIS")
                instr.write(":PAGE:CHAN:VMU1:DIS")
                instr.write(":PAGE:CHAN:VMU2:DIS")
                instr.write(":PAGE:CHAN:SMU1:VNAME 'VD' ")
                instr.write(":PAGE:CHAN:SMU2:VNAME 'VS' ")
                instr.write(":PAGE:CHAN:SMU1:INAME 'ID' ")
                instr.write(":PAGE:CHAN:SMU2:INAME 'IS' ")
                instr.write(":PAGE:CHAN:SMU1:MODE V")
                instr.write(":PAGE:CHAN:SMU1:FUNC VAR1")
```

```
                instr.write(":PAGE:CHAN:SMU2:MODE COMM")
                instr.write(":PAGE:CHAN:SMU2:FUNC CONS")

                instr.write(":PAGE:MEAS:VAR1:START 0")
                instr.write(":PAGE:MEAS:VAR1:STOP 5")
                instr.write(":PAGE:MEAS:VAR1:STEP 0.01")
                instr.write(":PAGE:MEAS:VAR1:COMP 0.01")
                instr.write(":PAGE:MEAS:VAR1:POINTS 7")

                instr.write(":PAGE:MEAS:SSTOP COMP")

                instr.write(":PAGE:SCON:SING")
                instr.write("*WAI")

                # Record time of measurement.
                cur_time = str(datetime.now().time())
                formatted_cur_time = cur_time[0:2] + cur_time[3:5] + cur
    _time[6:8]

                instr.write(":FORM:DATA ASC")

                I_data = instr.query_ascii_values(":DATA? 'ID' ")

                V_data = instr.query_ascii_values(":DATA? 'VD' ")

                # Section details writing the new found data to a file o
    n the computer.
                file = open("%s.txt" % formatted_cur_time, "w")

                file.write("\n")
                file.write("VD= 0 to 5 in 0.01 step\n")

                file.write("NO. VD ID\n")
                file.write(" V A\n")

                i = 1

                for x, y in zip(V_data,I_data):

                    file.write(str(i) + " " + str(intorfloat(x)) + "
     " + str(y) + "\n")

                    i = i + 1

                file.close()

                cur_time = None
                formatted_cur_time = None
```

```
                instr.write(“:PAGE:GLIS”)
                instr.write(“:PAGE:GLIS:SCAL:AUTO ONCE”)

                #*****************************************************************************************************#
                #*****************************************************************************************************#

                controller.wait(2500) # Wait 2.5 seconds before moving just to make sure the device is finished collecting data.

                NewportESP301Axis(controller,2).move(-0.08, absolute=False)
                NewportESP301Axis(controller,2).wait_for_stop()

                if col != devices_LtR-1:
                    NewportESP301Axis(controller,0).move(float(space_between_columns [row] [col]), absolute=False)
                    NewportESP301Axis(controller,0).wait_for_stop()

                NewportESP301Axis(controller,2).move(0.08,absolute=False)
                NewportESP301Axis(controller,2).wait_for_stop()

            NewportESP301Axis(controller,2).move(-0.08, absolute=False)
            NewportESP301Axis(controller,2).wait_for_stop()

            NewportESP301Axis(controller,0).move(-sum(space_between_columns [row]), absolute=False)
            NewportESP301Axis(controller,0).wait_for_stop()

            if row != num_of_rows-1:

                NewportESP301Axis(controller,1).move(-gap_between_rows[gap_counter], absolute=False)
                NewportESP301Axis(controller,1).wait_for_stop()
                gap_counter = gap_counter + 1

            elif row == num_of_rows-1:

                NewportESP301Axis(controller,1).move(sum(gap_between_rows), absolute=False)
                NewportESP301Axis(controller,1).wait_for_stop()

            NewportESP301Axis(controller,2).move(0.08,absolute=False)
            NewportESP301Axis(controller,2).wait_for_stop()

    else:
```

```
253.
254.         print ("PROBE CONTACT ERROR: Please attempt to make proper con
    tact with the device and try again.")
255.
256.         sys.exit()
257.
258. else:
259.
260.     print ("ALIGNMENT ERROR: Please attempt to align the sample proper
    ly by hand by adjusting the sample position and run the script again.")

261.
262.     sys.exit()
263.
264. print ("\nRun Completed.")
```

4.4 Example Results Text File

The following is the results text file for an I_{ds}–V_{ds} test on a HEMT:

```
1.
2.  VD= 0 to 10 in 0.1 step
3.  VG= -5 to 1 in 1 step
4.  NO. VD ID
5.   V A
6.  1 0 0.00018206
7.  2 0.1 0.00018476
8.  3 0.2 0.00018612
9.  4 0.3 0.00019228
10. 5 0.4 0.00019443
11. 6 0.5 0.0001943
12. 7 0.6 0.00019716
13. 8 0.7 0.00020149
14. 9 0.8 0.00019876
15. 10 0.9 0.00020181
16. 11 1 0.00020634
17. 12 1.1 0.00020678
18. 13 1.2 0.00020586
19. 14 1.3 0.00021149
20. 15 1.4 0.00021184
21. 16 1.5 0.00020952
22. 17 1.6 0.00021122
23. 18 1.7 0.00021476
24. 19 1.8 0.00021004
25. 20 1.9 0.00021208
26. 21 2 0.00021608
27. 22 2.1 0.00021551
28. 23 2.2 0.00021327
29. 24 2.3 0.00021831
```

```
30. 25 2.4 0.00021816
31. 26 2.5 0.00021462
32. 27 2.6 0.0002158
33. 28 2.7 0.00021889
34. 29 2.8 0.00021358
35. 30 2.9 0.00021504
1.      .
X.        .
X.        .
705. 700 9.3 0.0040164
706. 701 9.4 0.0040168
707. 702 9.5 0.0040184
708. 703 9.6 0.0040128
709. 704 9.7 0.0040118
710. 705 9.8 0.0040148
711. 706 9.9 0.0040129
712. 707 10 0.0040082
```

5. Conclusion

We successfully integrated the Micromanipulator P200L semiautomatic probe station with our 4155 semiconductor parameter analyzer to add automation to I-V testing of electronic devices such as diodes and HEMTs. The semiautomatic probe station will save time as well as allow for testing of a large number of devices on a given sample in a timely manner. This effort required extensive programming using the Python programming language, where this technical report will serve as a reference to both current and future users.

6. Future Work and Improvements

Future work and possible improvements are dependent on the user. The scripts described in this technical report are written in a way so that they are very adaptable to many different device types, geometries, and measurements. Different users are encouraged to generate new ideas that can be implemented into the scripts to improve upon it. In terms of current foreseeable future improvements, the Tektronix 370A programmable curve tracer can also be implemented into the system to complement the 4155C. The 370A is used for high-voltage measurements up to 2000 V. The 370A will have the exact same logic and process as the 4155C; the only difference is that the GPIB commands that are written to the 4155C will be replaced with a new set of commands with different syntax for the 370A. This new set of commands can be found in the 370A operator's manual.[4]

7. References

1. Agilent Technologies 4155C/4156C semiconductor parameter analyzer GPIB command reference. Santa Clara (CA): Agilent Technologies; 2004.
2. Casagrande S. InstrumentKit. 2016 [accessed 2016 Nov 29]. https://instrumentkit.readthedocs.io/en/latest/intro.html.
3. ESP301 integrated 3-Axis motion controller/driver user's manual. Irvine (CA): Newport; 2014.
4. Operator manual, 370A, programmable curve tracer. Beaverton (OR): Sony Tektronix; 1989.

List of Symbols, Abbreviations, and Acronyms

1-D	1-dimensional
2-D	2-dimensional
ARL	US Army Research Laboratory
GPIB	general-purpose interface bus
HEMT	high electron mobility transistor
I-V	current-voltage
UI	user interface
USB	universal serial bus

1 (PDF) DEFENSE TECHNICAL INFORMATION CTR DTIC OCA

2 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL CIO L IMAL HRA MAIL & RECORDS MGMT

1 (PDF) GOVT PRINTG OFC A MALHOTRA

1 (PDF) DIRECTOR US ARMY RESEARCH LAB RDRL SED E R TOMPKINS

INTENTIONALLY LEFT BLANK.